Using a Classifier Pool in Accuracy Based Tracking of Recurring Concepts in Data Stream Classification[†]

Mohammad Javad Hosseini, Zahra Ahmadi, Hamid Beigy
Sharif University of Technology
Tehran, Iran
{mjhosseini, z_ahmadi, beigy}@ce.sharif.edu

Abstract

Data streams have some unique properties which make them applicable in precise modeling of many real data mining applications. The most challenging property of data streams is the occurrence of "concept drift". Recurring concepts is a type of concept drift which can be seen in most of real world problems. Detecting recurring concepts makes it possible to exploit previous knowledge obtained in the learning process. This leads to quick adaptation of the learner whenever a concept reappears. In this paper, we propose a learning algorithm called Pool and Accuracy based Stream Classification (PASC), which takes the advantage of maintaining a pool of classifiers to track recurring concepts. Each classifier is used to describe an existing concept. Two methods are presented for classification task: active classifier and weighted classifiers methods. For the updating of the pool we use two methods: Bayesian and Heuristic. Experimental results on real and artificial datasets show the effectiveness of weighted classifiers method while dealing with sudden concept drifting datasets. In addition, the proposed updating methods outperform the existing algorithms in datasets with arbitrary attributes.

1 Introduction

As the data available on the web increases, processing the large volume of data and extracting knowledge from them is needed. These data are changing and they cannot be saved and processed wholly in the same way as classical data mining assumes. So, presenting new algorithms which could learn and classify using this continuous and unlimited stream of data is a challenging problem. Data streams have some properties [Tsymbal, 2004]:

- They could not be saved completely and so a forgetting mechanism is needed to forget ineffective data.
- The processing of data should be done online and the algorithm complexity should be simple.
- Most of the time, feature (or class) distribution is changed over the time. This is known as concept drift. If the drift takes effect in the target function, it is named real concept drift.

The concept drift could be sudden, gradual, incremental or recurring [Zliobaite, 2010a]. When the underlying distribution of data changes suddenly at time t_k, sudden drift occurs. Gradual drift happens when in a period of time, the data is drawn from two distributions and over time, the probability of the old distribution decreases and the probability of the new distribution increases. Incremental drift can be thought of as a generalized version of gradual drift. Here in the drift period, there could be more than two distributions to draw data from. However the difference between the distributions should be small. The other type of drift is recurring concept, where previously seen concepts reappear after some time. One important challenge in learning from data streams in the presence of concept drift is distinguishing the drift from the noise. It is important to note that I.I.D (Independent Identically Distribution) condition is not valid in the streams in which concept drift occurs, but it is rational to think that small size batches of data satisfy the I.I.D condition.

There have been extensive studies on sudden and gradual concept drift detection and learning [Baena-García et al., 2006; Gama and Castillo, 2006; Helmbold and Long, 1994; Klinkenberg and Joachims, 2000; Klinkenberg, 2004; Kolter and Maloof, 2007; Kuh et al., 1991; Gao et al., 2008; Nishida, 2008; Bifet et al., 2010a; Bifet et al., 2010b; Kuncheva and Zliobaite, 2009; Garnett, 2010; Ikonomovska et al., 2010; Scholz and Klinkenberg, 2006; Zliobaite, 2010b]. Early systems in data stream support recurring concepts [Schlimmer and Granger, 1986; Widmer and Kubat, 1993; Widmer and Kubat, 1996], however, they are mostly considered recently [Lazarescu, 2005; Gama and Kosina, 2009; Katakis et al., 2009; Morshedlou and Barforoush, 2009; Gomes et al., 2010], and identified as a challenging problem in data streams.

In this paper we propose a learning algorithm which tries to improve classifying concept drifting data streams by exploiting the existence of recurring concepts. This is done by maintaining a pool of classifiers which is updated continuously while processing consecutive batches of data (same as previous approaches, e.g. [Gomes et al., 2011; Katakis et al., 2009; Ramamurthy and Bhatnagar, 2007]). Each classifier of this pool is used to describe one of the existing concepts. When a new batch of data is received, first it is classified and after receiving the true labels of instances, it is used to update an existing classifier in the pool or add a new classifier to it. Deciding which classifier should be updated or whether a new one is needed is done by some examinations on the new batch of data and the pool. Classification of the instances is done by using the classifiers in the pool in an effective and adaptive way. This algorithm is similar to the one used in [Katakis et al.,

[†] This paper is the resubmission of a paper with the same topic published in Evolving Systems 4(1): 43-60 (2013). (an earlier version was published ICDM Workshops 2011).

2009], but there are major changes in the steps of the algorithm. In fact, our contribution is to propose a new method to classify instances called weighted classifiers method. The other novel part of the paper is the presentation of new methods to update the pool using Bayesian formulation and a heuristic method. Finally the presented methods are compared with the existing ones.

The results show the effectiveness of our algorithm in terms of accuracy and time especially in data streams of sudden drifts. In addition, it is tried to solve some parameter setting problems that exist in some of the previous methods.

The structure of this paper is as follows: in the next section the related works of recurring concepts is discussed. In section 3 the proposed algorithm is presented. Section 4 evaluates the proposed algorithm and compares the experimental results to some previous methods. Section 5 concludes the paper and discusses some future developments which can be done.

2 Related Work

Concept drift learning of data streams has been studied extensively in recent decades. As discussed previously, drifts can be of different types. Most of studies are done on the learning of sudden and gradual drifts. But one possible drift is the change of the current concept to one of the previously seen concepts. As in data streams the learner forgets some unused concepts passing the time, if instances from a previously seen concept is presented to the learner, it may classify them incorrectly. So the learner may be fallen into the trap. Recurring concepts detection and learning is a hard and challenging problem which has been studied in recent years [Lazarescu, 2005; Gama and Kosina, 2009; Katakis et al., 2009; Morshedlou and Barforoush, 2009; Gomes et al., 2011]. All of presented methods try to extract concept from received instances and maintain the concept in a pool of concepts. Every time a new instance arrives, the similarity to available concepts is measured and a model is selected or created. The rest of this section reviews the researches done in the area of recurring concepts in data streams.

The first algorithm supporting recurring concepts consists of an ensemble of classifiers [Ramamurthy and Bhatnagar, 2007]. Each classifier is built on a data chunk and none of the classifiers are deleted. Then while choosing classifiers for the ensemble, the algorithm selects only pertinent classifiers and so it supports the recurring concepts.

Reference [Katakis et al., 2009] presents a framework for the problem of recurring concepts. It extracts a conceptual vector from the arrived batch of data using a transformation function. We name the instances of a labeled batch as

$$\mathrm{B_L} = \{\boldsymbol{x}_{L(k)}, \boldsymbol{x}_{L(k+1)}, \dots, \boldsymbol{x}_{L(k+b-1)}\}, \quad (1)$$

where $\boldsymbol{x}_{L(k+i)}$ is the $(i+1)^{th}$ instance of the labeled batch of data. A conceptual vector $\boldsymbol{Z} = (z_1, z_2, \dots, z_n)$ is extracted from the batch where z_i is a conceptual feature and is calculated from

$$z_i = \begin{cases} \{P(f_i = v|c_j): i = 1..n, j = 1..m, v \in V_i\} & \text{if } f_i \text{ is nominal} \\ \{\mu_{i,j}, \sigma_{i,j} : j = 1..m\} & \text{if } f_i \text{ is numeric} \end{cases}, \quad (2)$$

where f_i is the i^{th} feature, V_i is the set of possible values of a nominal feature, $\mu_{i,j}$ and $\sigma_{i,j}$ are the mean and standard deviation of the j^{th} class of feature i. Then by using a clustering algorithm on the available concepts, the algorithm detects the recurring concepts. For each concept in the pool, the algorithm preserves a classifier which will be updated through the time. Clustering is done on the conceptual vectors and using the Euclidean distance as the similarity (difference) measure. If the similarity of a new conceptual vector is more than a threshold, an available concept and its classifier will be updated otherwise a new cluster and classifier will be created. One major problem of this framework is how to determine the threshold. The threshold value is a problem specific parameter and should be regularized by try and error.

Mean and standard deviation is used for the presentation of models in [Morshedlou and Barforoush, 2009] too. This approach uses a proactive behavior versus drifts: by knowing the current concept, it calculates the probability of next concept. If the probability is more than a threshold, the concept will be added to the buffer. If the algorithm detects a drift and decides to behave proactively, it selects a concept from the buffer. If the concept matches the batch, it will be updated. If the concept does not match the data and the algorithm behaves proactively, the next concept will be selected else if the reactive behavior is selected, a new classifier will be trained on the batch. [Morshedlou and Barforoush, 2009] uses a heuristic approach to select proactive or reactive action. Here a threshold parameter should be selected as well as doing some computations to select the suitable behavior each time which is a time consuming action.

The other approach uses meta-learners which can detect the reoccurrence of concepts and activate the previous classifiers using proactive behaviors [Gama and Kosina, 2009]. The meta-learner learns the space where the base learner does well. When the algorithm enters the warning phase of drift, meta-learners determine the performance of their corresponding base learners. If the performance is more than a threshold, the algorithm will use the base learner to classify next instances. Here all base learners and their corresponding meta-learners (referees) are maintained in the pool.

Another idea used in this domain is the use of context space model to extract concept from learning model [Gomes et al., 2010]. A context space is a *N*-tuple of the form $R = (a_1^R, a_2^R, \dots, a_N^R)$, where a_i^R determines the acceptable regions of feature a_i. Each classifier has a context space description and all of them will be saved in a repository. To select the appropriate model, the algorithm uses their corresponding contexts.

3 Proposed Learning Algorithm

Our goal is to propose a new method named *Pool and accuracy based Stream Classification (PASC)*. The idea followed in this method is similar to the method proposed in [Katakis et al., 2009]. We maintain a pool of classifiers which contains a number of classifiers each describing a particular concept which is being updated through the time. After receiving a batch of data, first we predict the labels of its instances and then receive the true labels. Then we can use the instances and their labels to update a classifier in the pool or create a new classifier on this batch of data and add it to the pool, if necessary. The classifiers added to the pool cannot grow arbitrarily the maximum number of classifiers in the pool cannot exceed a predefined limit which is a parameter of our algorithm.

To update or create a classifier in the pool, first of all the most relevant concept to the batch of labeled data is selected. If the similarity is more than a predefined threshold or the pool is full, we update the most relevant classifier with the newly arrived labeled batch. Otherwise we construct a new classifier on it. The classifiers used in our method can be any kind of updateable classifiers.

In the rest of this section, we seek how to classify the batches of data and update the pool. As mentioned above, after receiving each batch of data, the classification is done and after receiving their labels, we update the pool. In the proposed method, iteratively after receiving the t^{th} batch of unlabeled data $B_t = (x_{t,1},x_{t,2},\ldots,x_{t,k})$ such that $x_{t,i}$ is the i^{th} data of the t^{th} batch, and its labels $L_t=(l_{t,1},l_{t,2},\ldots,l_{t,k})$ such that $l_{t,i}$ is the label of $x_{t,i}$, we follow the general framework shown in Procedure 1.

Input: an infinite stream of batches of instances B_t.
After classification of each instance $B_{t,i}$, its label is revealed to the algorithm.
Output: Predicted labels of instances $B_{t,i}$.

```
1  Pool = Ø; // the pool of classifiers
2  C = make_classifier(B1,L1);
3  RDC = new classifier(); //only used in Bayesian
4  //method
5  ac = 1; // active classifier
6  W1=1;
7  Pool = Pool U {C};
8  X1=sum_data(B1);
9  RDC.update(X1,1); //1 is the label of X1
10 for j=2 to infinity do
11     Classify Bt.
12     Update Pool with Bt and Lt;
13     determine active classifier (classifier weights);
14 end for
```

Procedure 1. The main framework of PASC.

In line 2, *C* is the first classifier which will be added to the pool and W_1 (in line 6) is its weight. *RDC* is a classifier and *ac* contains the active classifier which will be used in the rest of the procedure. In line 8, X_1 is an instance constructed from B_1. This procedure contains three main phases which can be seen in lines 11 to 13.

In the following subsections, we consider the details of the parameters discussed above and the three phases of the algorithm.

3.1 Phase 1: Classifying the Batch

In this phase, after receiving a batch of unlabeled data B_t, we classify the batch using the classifiers in the pool. This task can be done in two ways. The first is similar to the method used in [Katakis et al., 2009] and the second tries to classify the batch using the weights assigned to the classifiers.

Classifying the Batch According to the Active Classifier

method is used in [Katakis et al., 2009] to classify instances using the classifiers in the pool. The classifier selected to classify the batch is named active classifier. This classifier is defined according to the last iteration. If in the last iteration, a new classifier was added to the pool, it would be the active classifier. Otherwise, the classifier that has the most relevance to the batch would be the active classifier. The pseudocode of this method is shown in Procedure 2. In line 2, *ac* is the active classifier and *pl* stores the predicted labels of the instances.

```
1  for i=1 to k do
2      pl[Bt,i] = Pool[ac].classify(Bt,i);
3  end for
```

Procedure 2. Classify batch according to active classifier.

Classifying the Batch According to the classifiers' weights.

The first way of classifying a batch uses the active classifier that is appropriate for the last batch of data. However, when a sudden concept drift occurs, the method's performance decreases significantly, because the appropriate classifier for the last batch is not appropriate for the current batch anymore. We suggest using the classifiers in the pool in an adaptive way. A positive weight is assigned to each classifier in the beginning of processing the batch according to the performance of the classifier on the previous batch and when we want to classify an instance, we use the classifier with the highest weight. When the true label is revealed to the algorithm, the classifiers' weights can be updated. Updating the weights is done according to the following rule:

$$w'(j) = w(j) * \beta^{M(j,i)}, \quad (3)$$

where $w(j)$ is the current weight of j^{th} classifier and $w'(j)$ is its new weight and β is a parameter in [0,1). If the j^{th} classifier classifies the i^{th} instance correctly, $M(j,i)$ will be 0, otherwise it is 1. Equation (1) is inspired from [Freund and Schapire, 1996] which models the online prediction problem with a two-player repeated game. The first player is the learner and the second is the environment. The leaner can choose a mixed strategy *P* that determines how to classify the instances determined by the mixed Strategy *Q* of the environment. The mixed strategy *P*, determines the weight of each concept to be used in the weighted majority method of classifying instances. The mixed strategy *Q* determines how to present instances to the learner. The game is as follows: First, the learner chooses mixed strategy *P* that determines how it would classify the instances, and then the environment chooses mixed strategy *Q* that determines how the instances are presented to the algorithm. In the next step, learner can observe the loss of using these strategies and so it can change its mixed strategy in the next iteration by updating the weights. It has been shown that for sufficient number of instances, the error of ensemble with the weights determined by (3) is sufficiently close to the best classifier's error [Freund and Schapire, 1996]. So if the size of the batch is large enough, the performance of our ensemble classifier on the current batch is close to the performance of the best classifier in the pool. But this size should not be so large that it violates the I.I.D condition in the batch or makes difficulty in storing data in the memory.

Although using this method is guaranteed to work well, we slightly modify the method to improve its efficiency. First, Instead of using weighted majority to classify an instance we use only the classifier with the highest weight. Second, Instead of applying the updating rule for every instance, we use it for a subsample of the batch that has the size equal to square root size of the batch.

The initial values of weights are 1 and after processing each batch, the weights are set according to the rule discussed in phase 3. The pseudocode of this method is

shown in Procedure 3. In line 1, S_t is a subsample of the batch B_t and m is its size which is set to the square root size of the batch. After classifying each instance in line 4, if the instance is a member of the subsample, classifiers' weights will be updated.

```
1   S_t=sub_sample(B_t,m);
2   /* makes a sub_sample of size m*/
3   for i=1 to k do
4       pl[B_t,i] = classifyw(Pool,W,B_t,i);
5       /*Uses the most weighted classifier*/
6       if S_t does not contains B_t,i
7           continue;
8       end if
9       for j=1 to size(Pool) do
10          W_j=W_j* Pool[j].error(B_t,i,L_t,i);
11      end for
12  end for
```

Procedure 3. Classify batch according to classifier weights.

3.2 Phase 2: Updating the Classifiers' Pool

After receiving L_t, the true labels of B_t, a classifier in the pool will be updated incrementally or a new classifier will be created on the batch. If we assume the size of the batch is small enough, it will be relevant to only one of the available concepts, because the concepts in the pool represent different hypotheses. So the relevant concept should be updated using the current batch of data. So we need to find the concept which describes B_t and L_t with the highest probability and also find a measure of its correspondence to the batch. In the following two subsections, two alternatives of performing this task are discussed. The first is a straightforward method and uses Bayes' theorem to find the probabilities. The second is a heuristic method which is more efficient than the first.

Bayesian method for Updating the Classifiers' Pool

In this method, we estimate the relevance probability of each available concept to B_t and L_t. As previously mentioned, in the environments subject to concept drift, the I.I.D condition does not hold. But we can assume that this condition holds for a batch of data that is sufficiently small. So the probability that B_t and L_t correspond to concept h_i can be formulated as:

$$P(h_i|B_t, L_t) = \frac{P(B_t, L_t|h_i) * P(h_i)}{P(B_t, L_t)}, \tag{4}$$

where the right side of the equation follows from Bayes' theorem. Thus the best concept to describe B_t and L_t is:

$$\begin{aligned} &argmax_i P(h_i|B_t, L_t) \\ &= argmax_i\ P(B_t, L_t|h_i) *\ P(h_i). \end{aligned} \tag{5}$$

Equation (5) uses the fact that the best concept does not relate to the probability of B_t and L_t. As the environment is non-stationary and we cannot have any assumption about the concepts, we consider $P(h_i)$ which is the prior probability of the i^{th} concept to be identical for all concepts. So equation (5) becomes:

$$\begin{aligned} &argmax_i P(h_i|B_t, L_t) \\ &= argmax_i\ P(B_t, L_t|h_i) \\ &= argmax_i P(B_t|h_i) * P(L_t|B_t, h_i). \end{aligned} \tag{6}$$

Hence we should estimate $P(L_t|B_t,h_i)$ and $P(B_t|h_i)$. The former is the conditional probability that the labels of the instances $(x_{t,1},x_{t,2},\ldots,x_{t,k})$ be $(l_{t,1},l_{t,2},\ldots,l_{t,k})$ given that the instances and their labels are described by the i^{th} concept and the latter is the probability that the batch is produced in an environment described by the i^{th} concept.

According to I.I.D condition in a batch, we have:

$$P(L_t|B_t, h_i) = \prod_{j=1}^{j=k} P\left(l_{t,j}|x_{t,j}, h_i\right). \tag{7}$$

Notice that $P(l_{t,j}|x_{t,j},h_i)$ can be estimated using the posterior probability calculated by the i^{th} classifier. To estimate $P(B_t|h_i)$, according to I.I.D we have:

$$P(B_t|h_i) = \prod_{j=1}^{j=k} P\left(x_{t,j}|h_i\right). \tag{8}$$

There is a straightforward way to determine $P(x_{t,j}|h_i)$ by using a classifier which we call *raw data classifier*. The input of this classifier is the unlabeled instances $x_{t,j}$ and its output is the probability of the instances to belong to the concepts. So to train the raw data classifier, first the concept which describes B_t and L_t best, is determined. Then all instances in the batch and the concept index (or its id) as the class label are given to the classifier to be updated. To determine the relevant concept of the batch, we can give all of the batch instances to the classifier. But this will take much time to find $P(B_t|h_i)$ and therefore we use an alternate way: instead of using all instances in the batch we make an instance X_t for the batch B_t and use it to train raw data classifier (RDC). X_t has the same number of features as the original instances and its i^{th} feature is simply the sum of all the i^{th} features of the instances in the batch.

After receiving unlabeled batch B_t, X_t is built and the probability of each of its instances to belong to any of the concepts in the pool is estimated by the probability of X_t to belong to the concept which can be calculated by RDC. Then the best concept matching B_t and L_t is determined (it may be a new concept added to the pool) and X_t and the best concept index are given to RDC to be updated. So $P(B_t|h_i)$ can be estimated as:

$$P(B_t|h_i) = {p_i}^k, \tag{9}$$

where p_i is the probability of belonging X_t to i^{th} concept which is calculated by RDC. Therefore, to determine the best concept describing B_t and L_t we can use:

$$\begin{aligned} &argmax_i P(h_i|B_t, L_t) \\ &= argmax_i\ {p_i}^k * \prod_{j=1}^{j=k} P\left(l_{t,j}|x_{t,j}, h_i\right). \end{aligned} \tag{10}$$

To prevent underflow of the products we use (11) Instead of (10) to find the best concept:

$$\begin{aligned} &argmax_i P(h_i|B_t, L_t) \\ &= argmax_i\ k * log\ p_i + \sum_{j=1}^{j=k} log\ P\left(l_{t,j}|x_{t,j}, h_i\right). \end{aligned} \tag{11}$$

If the pool is not full and the result of the expression computed in (11) is less than a parameter θ_1, a new classifier will be added.

Using this method, we must find the posterior probability of k instances for finding the best concept and this will take much time. To resolve this problem, relying on the fact that the instances in the batch are I.I.D, only a sub-

sample of the square root size of the batch is used to estimate the best concept. The pseudocode of this method is shown in Procedure 4. In line 2, S_t contains a subsample of the batch B_t and m is its size which is set to the square root size of the batch. SL_t stores the labels of S_t. Lines 5 to 7 find the best describing classifier of the batch according to Bayesian method. The variable *bestC* refers to the best classifier and *maxA* indicates the result of the expression computed in (11) for *bestC*.

```
1   X_t = sum_data (B_t);
2   S_t = sub_sample (B_t,m);
3   SL_t = sub_sample (L_t,m);
4   /* stores the labels of the S_t*/
5   (maxA , bestC) = (max,argmax)_{j:1..size(Pool)}
6   (m* log (RDC.prob(x_t,j)) +
7   Σ_{i=1:m} log (Pool[j].prob(S_i,SL_i)) );
8   if (maxA>θ_1 or size(Pool)>maxC)
9       Pool[bestC].update(B_t,L_t);
10  else
11      C = make_classifier(B_t,L_t);
12      Pool = Pool U {C};
13      bestC = size(Pool);
14  end if
15  RDC.update(x_t,bestC);
```

Procedure 4. Bayesian method for updating classifiers' pool.

Heuristic method for Updating the Classifiers' Pool

To find the best concept describing B_t and L_t, the accuracy of all classifiers on B_t will be measured. If the pool is full and a new classifier cannot be added, the best classifier is updated with B_t and L_t. But if the pool is not full and the accuracy of the best classifier for this batch of data is more than a parameter θ_2, then the best existing classifier is updated by B_t and L_t. Otherwise if the accuracy of classifier is less than θ_2, a new classifier is created and trained on this batch. The reason of using this approach is that the more the accuracy of a classifier on the current batch is, the more relevance it may have to the batch. Therefore, the concept this classifier describes can be refined or extended using the current batch of data. The pseudocode of this method is shown in Procedure 5. Lines 4 and 5 find the best classifier describing the batch according to heuristic method. The variable *bestC* refers to the best classifier and *maxA* indicates the accuracy of that classifier on the current batch.

```
1   S_t = sub_sample (B_t,m);
2   SL_t = sub_sample (L_t,m);
3   /* stores the labels of the S_t*/
4   (maxA , bestC) = (max,argmax)_{j:1..size(Pool)}
5   (pool[j].accuracy(S_t,SL_t));
6   if (maxA>theta or size(Pool)>maxC)
7       Pool[bestC].update(B_t,L_t);
8   else
9       C = make_classifier(B_t,L_t);
10      Pool = Pool U {C};
11      bestC = size(Pool);
12  end if
```

Procedure 5. Heuristic method to update classifiers' pool.

3.3 Phase 3: determining the active classfier (or classfier weiths)

After phases 1 and 2 are done, some final operations should be done before moving to the next iteration. If phase 1 is done according to the active classifier, the active classifier should be set. Active classifier is the one that has been updated with the current batch of data, i.e. the bestC parameter of our algorithm.

If phase 1 is done in the second way, the weights should be initialized for the next iteration. The weights of the classifiers in the pool are set so that in the next iteration, the performance of the method will be high. Each classifier is tested on a subsample of the square root size of the batch and its weight is set by:

$$w_0(i) = \beta^{(2^{A(i)})}, \tag{12}$$

Where *A(i)* is the accuracy of the i^{th} classifier. A classifier which classifies the current batch poorly, will have a less initial weight. Some kind of locality assumption is used in (12) for setting the initial weights which does not work properly when a sudden concept drift occurs. Phase 1 tries to handle this problem by updating the weights while processing the batch. The pseudocode of this method is shown in Procedure 6.

```
1   S_t = sub_sample (B_t,m);
2   SL_t = sub_sample (L_t,m);
3   for j=1 to size(Pool) do
4       c_error= Pool[j].error(S_t,SL_t);
5       W_j= beta^(2^c_error);
6   end for
```

Procedure 6. Determine classifier weights.

4 Experimental Results

In this section, we first introduce the data sets containing recurring concepts which are used in the experiments. Then we discuss the parameter tuning of our method and compare it to the parameters of CCP framework. In the last subsection the proposed methods are compared with each other and the CCP framework, one of the most promising frameworks developed in the tracking of recurring concepts. The experiments show the effectiveness of our method.

4.1 Data sets

Three real datasets and one artificial dataset are chosen for the experiments given in this section. The artificial dataset is moving hyperplanes and contains sudden concept drift. Real datasets are emailing list [Katakis et al., 2009], spam filtering and sensor data. Emailing list and spam filtering are high dimensional datasets and sensor data is a very large real dataset. Emailing list and hyperplane datasets contain sudden concept drift and spam filtering and sensor data contain gradual drift.

Emailing List Dataset

The emailing list (elist) dataset which is used in [Katakis et al., 2009] contains a stream of emails about different topics shown to the user one after another and are labeled as interesting or junk. To construct this dataset, the data in usenet posts [Frank, 2010] which exists in 20 newsgroups collection is used and three topics are selected. The user is interested in one or two topics in each concept and so he/she labels the emails according to his/her interest. The interests of the user can be changed in time and so this dataset simulates recurring concepts and concept drift (Table 1). The dataset contains 1500 instances with 913 attributes and is divided into 5 time periods with equal number of instances .

Spam Filtering Dataset

This dataset is obtained from Spam Assassin[‡] collection and contains email messages. The dataset consists of 9324 instances with 500 attributes and represents gradual concept drift .

Table 1. Emailing List Dataset (elist) [Katakis et al. 2009]

	1-300	300-600	600-900	900-1200	1200-1500
Medicine	+	-	+	-	+
Space	-	+	-	+	-
Baseball	-	+	-	+	-

Hyperplane Dataset

This dataset simulates the problem of predicting class of a rotating hyper plane. In an *n*-dimensional space, a hyper plane decision surface is the equation $g(\vec{x}) = \vec{w}.\vec{x} = 0$ where $\vec{w}$ determines the orientation of the surface and $\vec{x}$ is an instance in the space. If $g(\vec{x}) > 0$, $\vec{x}$'s label is 1, otherwise it is 0. To simulate concept drift, the orientation of the hyper plane is changed over time. Our dataset has 8000 instances with 30 real attributes. There is a concept drift after each 2000 instances. There are only two concepts which reappear after the first 4000 instances. This dataset shows the problem of sudden concept drift and recurring concepts.

Sensor dataset

Sensor dataset is a real dataset which consists of the information collected from 54 sensors deployed in Intel Berkeley Research laboratory in a two-month period [Zhu, 2010]. The class label is the sensor ID, so there are 54 classes, 5 attributes and 2,219,803 instances. The type and place of concept drift is not specified in the dataset but it is obvious that there are some drifts. For example, lighting or the temperature of some specific sensors during the working hours is much stronger than nights or weekends

4.2 Parameter Tuning

One of the advantages of the proposed method is that its parameters can be tuned in a much simpler way compared to the CCP framework method and small changes of parameter values, do not lead to major variations in performance. On the other hand, the CCP framework method has a θ parameter which is somehow similar to our θ_1 and θ_2 parameters. If this parameter is set wrongly in CCP framework method, the accuracy of the classification will decrease significantly. For example, θ should be 4 for elist and 2.5 for spam filtering dataset. If we set θ to 2.5 instead of 4 for elist dataset, its accuracy will be 55% rather than 77%.

If weighted classification method is used in phase 1, a parameter β is required to update the weights which is by definition in [0,1). The more sudden the concept drift is, the smaller the parameter should be. We have set this parameter to 0.1 for all datasets. Another parameter is the maximum classifier number (*maxC*) which is set to 10 and implies that we expect to have at most 10 different concepts. In addition, we have a parameter θ_1 in the heuristic method which is a threshold for the accuracy of the best classifier. So the more the *maxC* parameter is and the less sudden the concept drift is, the higher θ_1 should be. We have set this parameter to 0.95 for all datasets which means that only when a classifier has the accuracy more than 0.95 on a batch, it will describe the concept of the batch correctly. For the other parameter, θ_2, in the Bayesian method, we have set it to $2 * \log(0.75) * m$, according to its definition. This is because we believe if each of the *2m* probabilities of (11) is at least 0.75, then the concept can be relevant to the batch and its labels. The batch size is set to 50 for elist and spam filtering datasets and 500 for hyperplane dataset.

As a result, parameter tuning for our method is simpler than CCP framework method and the same parameters work well for all datasets with different natures we have chosen. The only parameter that does not have the same value for all datasets in our experiments is the batch size. This problem also exists in the CCP Framework method and must be resolved according to the properties of the dataset.

The reason behind our claim that our parameter setting is simple is that most of these parameters can be expressed as some property of the datasets, but setting the parameters correctly needs some knowledge about the dataset.

4.3 Results and Discussion

We compared our method with the CCP Framework method [Katakis et al., 2009] in terms of accuracy, precision, recall and running time. We have discussed how to tune our method's parameters in the previous subsection. The results of our experiments on elist, spam filtering, hyperplane and sensor datasets are shown in tables 2, 3, 4 and 5, respectively.

comparison of methods' accuracies, precisions and recalls

The results for elist and hyperplane datasets that simulate sudden concept drift are much better when using the weighted classifiers method rather than active classifier method. The difference of about 8% in the accuracies can be seen. We have tested the weighted classifiers method in conjunction with the CCP framework method and the same result can be seen in terms of increase in the accuracy. This is reasonable, because when a sudden concept drift occurs, the active classifier which is appropriate for the last batch works poorly in classifying the current batch. When the weighted classifiers method is used, after receiving the first few instances of the batch, the classifier' weights are adapted so that the concept drift is taken into account and the classification task will have a higher accuracy.

As a comparison, our weighted classifiers method outperforms the CCP framework method for sudden concept drift and has similar results for gradual concept drift. Our batch assignment methods (Bayesian and heuristic) have results similar to the CCP framework method without having parameter setting problems discussed previously.

In sensor dataset, CCP and Bayesian batch assignment methods have lower performances (between 9% and 15% of accuracy) than Heuristic method. This means that CCP framework and Bayesian method have some problems in determining the true concept of a batch in sensor dataset. One problem with CCP framework method is that it uses the Euclidean distance as the measure of similarity of a batch to a concept. ConDis, the distance measure used in CCP, is dependent on the magnitude of the attribute values and an attribute with large values can reduce the effects of the other attributes in the distance calculation. The problem of Bayesian method could be possibly the I.I.D as-

[‡] The Apache SpamAssasin Project - http://spamassassin.apache.org/

sumptions made in it. However, Bayesian method still outperforms than CCP framework method (about 3%).

Table 2. Results of all methods on elist dataset.

Batch assignment Method	Classification Method	Acc.	P	R	F-measure	Time
CCP	Active	0.77	0.73	0.81	0.77	1004
	Weighted	0.82	0.79	0.83	0.81	1274
Heuristic	Active	0.75	0.71	0.77	0.74	1816
	Weighted	0.82	0.8	0.83	0.81	1843
Bayesian	Active	0.75	0.71	0.8	0.75	2089
	Weighted	0.82	0.8	0.84	0.82	2462

Table 3. Results of all methods on spam filtering dataset.

Batch assignment Method	classification method	Acc.	P	R	F-measure	Time
CCP	Active	0.91	0.91	0.84	0.94	2217
	Weighted	0.89	0.92	0.87	0.93	2820
Heuristic	Active	0.89	0.91	0.84	0.93	3942
	Weighted	0.89	0.92	0.89	0.93	4112
Bayes	Active	0.89	0.9	0.86	0.93	4537
	Weighted	0.88	0.91	0.91	0.92	5405

Table 4. Results of all methods on Hyperplane dataset.

Batch assignment Method	classification method	Acc.	P	R	F-measure	Time
CCP	Active	0.76	0.72	0.81	0.78	868
	Weighted	0.83	0.81	0.83	0.84	947
Heuristic	Active	0.76	0.73	0.77	0.78	974
	Weighted	0.84	0.81	0.83	0.85	970
Bayes	Active	0.78	0.75	0.8	0.8	876
	Weighted	0.86	0.83	0.84	0.87	899

Table 5. Results of all methods on sensor dataset.

batch assignment	classification method	Accuracy	Time
CCP	Active	0.71	370560
	Weighted	0.71	813398
Heuristic	Active	0.87	929289
	Weighted	0.86	846226
Bayes	Active	0.74	883682
	Weighted	0.74	1299652

Comparision of methods' run times

The run time of each method is shown in the last column of the result tables (Table 2-5). The most time consuming part of these methods is the time spent calling the training and test methods of the classifiers. In the CCP framework method additional time is spent on the construction of the conceptual vectors and the clustering task. In all methods, each instance of the batch is used once to update a classifier in the pool. The difference is in the number of times an instance is classified or its posterior probability distribution is measured by the classifiers. Simply, assume that T_0 is the time taken to classify an instance and T_1 is the time taken to find the posterior probabilities for it. In the classification task, each data is classified only once in all batch assignment methods and so the only major differences are in updating the classifiers' weights and in phase 2 where the updating of the classifiers' pool is done. Suppose that the subsample size of the batch used in both the heuristic and the Bayesian methods is m. In the heuristic method, each of the m instances is classified once using all of the classifiers in the pool and in the Bayesian method, the posterior probabilities of each of the m instances are measured by each of the classifiers. In the Bayesian method, one posterior probabilities estimation and one update by the raw data classifier is also required for each batch but this can be ignored. So the time required in the heuristic method is at most $m * maxC * T_0$ and in the Bayesian method is at most $m * maxC * T_1$. T_1 is greater or equal to T_0 according to their definitions. So in general, we expect using the Bayesian method is more time consuming rather than the heuristic method, because the maximum time computed for Bayesian method is greater. This can be seen in tables 2 and 3, but not in the last dataset, because in this problem setting only two classifiers are added to the pool for the Bayesian method (among 10 possible classifiers).

In addition, we use a subsample of the batch to update the weights in the weighted classifiers method. Each of the instances in this subsample is classified by each of the classifiers in the pool to find the classifiers' errors. So if we use the same subsample of the batch for both updating the classifiers and their weights, we will obtain a time saving when using Heuristic and weighted classifiers methods. Therefore for each of batch assignment methods, using weighted classifiers method will consume more time than using active classifier. This can be seen in tables 2 to 4 for our three datasets, except in the Heuristic method because of the time saving we mentioned.

At last, Bayesian method takes the most time among all batch assignment methods while Heuristic and CCP methods take almost the same time using active classifier and Heuristic method is better when using weighted classifiers.

5 Conclusion and Future Works

We have proposed a method with some variations for streaming data classification in the presence of concept drift and recurring concepts. The general framework used in this paper maintains a pool of classifiers and updates them according to consecutive batches of data. The classifiers in the pool are used to classify new batches of data. The most similar method to our method is the CCP framework. Our method improves the accuracy while its parameter tuning is simpler.

Some future research works related to this study might include the followings. First, managing the classifiers in the pool can be done more complexly. For example, classifiers can be merged or removed to handle more complicated situations. Second, parameters of the algorithm are

dependent on the datasets. If they can be set dynamically according to the datasets, the algorithm will work properly for all datasets. Third, the algorithm should be run on more real datasets in order to achieve more reliable results.

References

[Baena-García et al., 2006] Manuel Baena-García, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavaldà and Rafael Morales-Bueno, Early Drift Detection Method, in ECML PKDD Workshop on Knowledge Discovery from Data Streams. 2006.

[Bifet, 2009] Albert Bifet, Adaptive Learning and Mining for Data Streams and Frequent Patterns, in Departament de Llenguatges i Sistemes Informatics. 2009, Universitat Politecnica de Catalunya.

[Bifet et al., 2010a] Albert Bifet, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Accurate ensembles for data streams: Combining restricted Hoeffding trees using stacking. in 2nd Asian Conference on Machine Learning. 2010. Tokyo, Japan: JMLR.

[Bifet et al., 2010b] Albert Bifet, A., Geoff Holmes, Richard Kirkby, Bernhard Pfahringer, MOA: Massive Online Analysis. Journal of Machine Learning Research. 2010, **99**: pp. 1601-1604.

[Frank, 2010] Frank, A., UCI Machine Learning Repository. 2010. accessed on May 2011; Available from: http://archive.ics.uci.edu/ml.

[Freund and Schapire, 1996] Yoav Freund, and Robert .E. Schapire. Game theory, on-line prediction and boosting. in Proceedings of the ninth annual conference on Computational learning theory. 1996: ACM.

[Gama and Castillo, 2006] Joa Gama and Gladys Castillo, Learning with local drift detection, in Advanced Data Mining and Applications, Proceedings, X. Li, O.R. Zaiane, and Z.H. Li, Editors. 2006, pp. 42-55.

[Gama and Kosina, 2009] Joa Gama and Petro Kosina, Tracking Recurring Concepts with Meta-learners, in Proceedings of the 14th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence. 2009.

[Gao et al., 2008] Jing Gao, Bolin Ding, Wei Fan, Jiawei Han, Classifying Data Streams with Skewed Class Distributions and Concept Drifts. IEEE Internet Computing, 2008. **12**(6): pp. 37-49.

[Garnett, 2010] Roman Garnett, Learning from Data Streams with Concept Drift, in Department of Engineering Science. 2010, University of Oxford. pp. 163.

[Gomes et al., 2011] Joao B. Gomes, Ernestina Menasalvas, and Pedro A.C. Sousa, Learning recurring concepts from data streams with a context-aware ensemble, in Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 994-999.

[Helmbold and Long, 1994] David P. Helmbold and P. M. Long, Tracking Drifting Concepts by Minimizing Disagreements. Machine Learning, 1994, 14, pp.27-45.

[Ikonomovska et al., 2010] Elena Ikonomovska, Joa Gama, and S. Deroski, Learning model trees from evolving data streams. Data Mining and Knowledge Discovery, 2010. **23**(1): pp. 128-168.

[Katakis et al., 2009] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas, Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowledge and Information Systems, 2009. **22**(3): pp. 371-391.

[Klinkenberg and Joachims, 2000] Ralf Klinkenberg and Thorsten Joachims, Detecting Concept Drift with Support Vector Machines. In the Proceedings of the Seventeenth International Conference on Machine Learning (ICML), 2000, pp.487-494.

[Klinkenberg, 2004] Ralf Klinkenberg. Learning Drifting Concepts:Example Selection vs. Example Weighting. Intelligent Data Analysis, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift, 2004, **8**(3), pp.281-300.

[Kolter and Maloof, 2007] J. Zico Kolter and Marcus A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts. Journal of Machine Learning Research, 2007. **8**: pp. 2755-2790.

[Kuh et al., 1991] A. Kuh, T. Petsche and H. Rivest, Learning Time-Varying Concepts. In Advances in Neural Information Processing Systems (NIPS), 1991, pp.183-189.

[Kuncheva and Zliobaite, 2009] Ludmila I. Kuncheva and Indre Zliobaite, On the window size for classification in changing environments. Intell. Data Anal., 2009. **13**(6): pp. 861-872.

[Lazarescu, 2005] Mihai M. Lazarescu, A Multi-Resolution Learning Approach to Tracking Concept Drift and Recurrent Concepts, in 5th IAPR Workshop on Pattern Recognition in Information Systems (PRIS). 2005: Miami, USA. pp. 52-61.

[Morshedlou and Barforoush, 2009] Hossein Morshedlou, and Ahmad A. Barforoush, A New History Based Method to Handle the Recurring Concept Shifts in Data Streams. World Academy of Science, Engineering and Technology, 2009. **58**: pp. 917-922.

[Nishida, 2008] Kyosuke Nishida, Learning and Detecting Concept Drift, in Information Science and Technology. 2008, Hokkaido University: Hokkaido.

[Ramamurthy and Bhatnagar, 2007] Sasthakumar Ramamurthy and Raj Bhatnagar. Tracking Recurrent Concept Drift in Streaming Data Using Ensemble Classifiers. in Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA '07). 2007. pp. 404-409.

[Scholz and Klinkenberg, 2006] Martin Scholz and Ralf Klinkenberg, Boosting Classifiers for Drifting Concepts. Intelligent Data Analysis, Special Issue on Knowledge Discovery from Data Streams, 2007, **11**(1), pp.3-28.

[Schlimmer and Granger, 1986]Jeffrey C Schlimmer and Richard H Granger Jr, Incremental learning from noisy data. Machine learning,1986, 1(3), pp. 317-354.

[Widmer and Kubat, 1993] Gerhard Widmer and Miroslav Kubat, Effective learning in dynamic environments by explicit context tracking. in Proceedings of the European Conference on Machine Learning, 1993, pp.227-243.

[Widmer and Kubat, 1996] Gerhard Widmer and Miroslav Kubat, Learning in the Presence of Concept Drift. Machine Learning, 1996, 23, pp.69-101.

[Witten et al., 2005] Ian H. Witten, Eibe Frank, Mark A. Hall, Data Mining: Practical machine learning tools and techniques. 2005: Morgan Kaufmann.

[Tsymbal, 2004] Alexey Tsymbal, The Problem of Concept Drift: Definitions and Related Work. 2004.

[Zliobaite, 2010a] Indre Zliobaite, Learning under Concept Drift: an Overview. 2010.

[Zliobaite, 2010b] Indre Zliobaite, Adaptive Training Set Formation. 2010, Vilnius University.

[Zhu, 2010]Xingquan Zhu, *Stream Data Mining repository*.2010. Accessed on Jan 2012; Available from: http://www.cse.fau.edu/~xqzhu/stream.html